

James W. Huffman  
1832 N. Cascade Ave.  
Colorado Springs, CO 80907  
719.475.7103  
719.623.0141 fax  
jim@huffmanlaw.net

BOUNDARY ADDRESS  
REGISTERS FOR SELECTION OF ISA MODE

by


Michael Gottlieb Jensen

Morten Stribaek

5

---

CROSS-REFERENCE TO RELATED APPLICATIONS

*Sub AI*  This application is related to copending U.S. Patent Application Serial Number \_\_\_\_\_ (Docket: MIPS:0102.00US), filed on \_\_\_\_\_, entitled

10 *Translation Lookaside Buffer for Selection of ISA Mode*, by common inventors, and having the same assignee as this application.

BACKGROUND OF THE INVENTION

*1. Field of the Invention*

15 This invention relates in general to the field of instruction processing in computer systems, and more particularly to an apparatus and method in a CPU for executing application programs that consist of program instructions belonging to different instruction set

20 architectures.

000001-2120260

## 2. Description of the Related Art

A first-generation computer was only capable of executing programs that were encoded using a unique set of programming instructions. The unique set of programming instructions, or instruction set architecture (ISA) was to be used to develop application programs for execution only on that first-generation computer. Because of this constraint, system designers typically selected a particular computer for use as a system central processing unit (CPU) based upon its hardware characteristics (e.g., speed, power consumption, etc.) in conjunction with its instruction set's ability to implement certain critical functions within a system design. Once the CPU was selected, the system application programs were developed using instructions from the CPU's instruction set and the application programs were exclusively executed on the selected CPU. If system designers desired to upgrade the system's CPU to a more powerful processor, then they were required to recode the system application programs using instructions from the instruction set of the more powerful processor. In the early days of software engineering, this was not a significant encumbrance, primarily because there were not very many application programs in existence, and those that had been developed were not very complex.

Sub  
A2

Because a CPU can be easily programmed to perform a wide variety of functions within a system design, within just a few years the number of CPUs and application programs in the marketplace increased exponentially. In parallel with these events, technological advances in the integrated circuit design and fabrication arts began to release a steady stream of more powerful and complex CPU designs. And as these more powerful and complex CPU designs were exploited, a number of modification and upgrade mistakes were made as a result of recoding existing application programs. So, hardware and software designers were required to focus on preserving and reusing a substantial amount of code that had already been developed and tested for use particular CPU designs. Consequently, as newer CPUs were introduced, in addition to implementing a whole "new" set of instructions, the CPUs retained the capability to execute applications that were coded with "old" instructions. Typically, this ability to execute multiple instruction sets was bounded by a particular manufacturer's line of products. For example, Digital Equipment Corporation produced a VAX11 CPU that supported newer VAX11 instructions and older PDP11 instructions.

Today, the number of application programs and their complexity continues to grow. In addition to this growth, another factor has provided both a motivation for innovation

as well as a cause for concern. That is, the number and diversity of instructions sets that are available today for use in programming applications has resulted in designers often first choosing a specific instruction set for  
5 implementation of a system design. Following this selection, one of many CPUs is selected that implements the specific ISA. In fact, many present day processors implement more than one ISA. These processors are also capable of executing an application program consisting of  
10 program modules that are coded by instructions from different ISAs, i.e., a multiple-ISA application program. Accordingly, a system designer can specify a specific ISA for encoding a specific set of program functions (e.g., signal processing algorithms) and select other ISAs for  
15 encoding other types of program functions (e.g., operating system functions, I/O functions, general purpose functions).

Program instructions are represented as binary values. When a particular program instruction is fetched from memory and provided to a multiple-ISA CPU for execution, the CPU  
20 must have some way of knowing which set of instruction decoding rules to apply in order to correctly process a program instruction that has been fetched from a multiple-ISA application program.

One approach to indicating the ISA mode for program instructions is to encode the ISA mode as an additional field of the instruction. But this approach is very memory inefficient because additional memory bits are required for each instruction in a program. A more workable approach, employed by present day multi-ISA CPUs, recognizes the fact that adjacent program instructions in a multiple-ISA application program tend to be from the same ISA. Hence, the technique that is used today to indicate the ISA mode of particular instruction streams is to insert a special program instruction into the instruction stream that directs the CPU to switch ISA modes when instructions from a different ISA are programmed. For example, when a CPU is executing a program module consisting of ISA 1 instructions, and the module wishes to transfer program control to a subroutine comprised of ISA 3 instructions, prior to transferring control to the subroutine, an ISA 1 mode switch instruction must be executed that directs the CPU to switch to ISA 3 mode. Following this, program control is transferred to the subroutine that consists of ISA 3 instructions.

The technique described above comes in various forms. Hammond et al., in U.S. Patent number 5,638,525 and U.S. Patent number 5,774,686, discusses a "switch" instruction that directs a multi-ISA CPU to perform an ISA mode switch

and to transfer program control. Jaggar, in U.S. Patent number 5,568,646 and U.S. Patent number 5,740,461, discusses the use of mode bits within an internal CPU register to signal a specific ISA mode. Under Jaggar's approach, a calling module first executes an instruction to set the mode bits in the internal CPU register to indicate the ISA mode of a module that is to be called. Following this, control is transferred to the called module. Nevill, in U.S. Patent number 5,758,115, and U.S. Patent 6,021,265, describes the use of predetermined indicator bits within a program counter register for signaling ISA modes. The program counter register within a CPU carries both the address for the instruction that is to be fetched from memory and the predetermined bits indicating the ISA mode of the instruction.

All of the above techniques have one shortcoming in common: there is an interdependency that exists between components of a multi-ISA application program that extends beyond the simple transfer of program control. More specifically, a transferring component must know the particular ISA mode of a component to which flow is to be transferred in order to direct the CPU to switch ISA modes. One skilled in the art will appreciate that this is a difficult approach for use in a complex application program environment because each time a given component of a

multiple-ISA application program is encoded into a different ISA mode, it forces a designer to modify all of the components that are referenced by the given component as well, thus increasing the chances for bugs to enter into a system design.

Sub A3  
Years ago however, Larson, in U.S. Patent number 5,115,500, proposed an approach for enabling a CPU to switch ISA modes during the execution of a multiple-ISA application program that did not require the insertion of a mode switch instruction into the flow of a transferring component. Larson associated a program instruction's address in the CPU's address space with one of several ISA modes. In essence, Larson used the upper bits of the program instruction's address to indicate its ISA mode. Hence, all instructions corresponding to a specific ISA mode were stored in one or more memory segments that corresponded to that specific ISA mode. Although Larson's technique addressed the issue of inserting mode switch instructions into an application program, his technique for using the upper bits of a fetched instruction's address as an indication of the instruction's ISA mode is restrictive because it requires that the CPU's address space be partitioned into fixed and equal-sized segments. And fixed, equal-sized segments do not represent the distribution of components according to different ISA modes within a



Cont  
A3

multiple-ISA application program. Larson's technique for switching ISA modes is inflexible and memory inefficient.

Therefore, what is needed is an apparatus that enables a multiple-ISA CPU to select a particular ISA mode for processing a particular program instruction that does not employ fixed and inflexible segments within the CPU's address space.

In addition, what is needed is an ISA mode selection apparatus that provides for execution of a multiple-ISA application program, where a given component of the application program can be modified to a different ISA mode without requiring that all components referenced by the given component be modified as well.

Furthermore, what is needed is an apparatus for executing a multiple-ISA application program on a CPU that eliminates the need to insert special mode switch instructions into the flow of a first component of the application program in order for the first component to transfer program control to a second component that is encoded by instructions from a different ISA mode.

Moreover, what is needed is a method for executing multiple-ISA application programs that reduces the number of changes required to the application program when one of its

subcomponents is modified to employ instructions from a different ISA.

### SUMMARY

5 The present invention provides a technique for encoding and executing multiple-ISA application programs that gives system designers the flexibility to dynamically configure the address space of a multiple-ISA CPU to meet the unique ISA mode storage requirements of components within the programs. In addition, the present invention obviates the need for inserting special mode switch instructions into the program flow of the application programs to effect a mode switch during their execution. Furthermore, the present invention advantageously allows designers to independently change a particular component of the application program to a different ISA without requiring that they modify all of the components that are referenced by the particular component as well.

20 In one embodiment, Instruction Set Architecture (ISA) selection logic within a CPU is provided for selecting an ISA decoding mode corresponding to a program instruction, where the program instruction is located at an address within an address space of the multiple-ISA CPU. The selection logic includes a plurality of boundary address

registers and ISA mode selection logic. The plurality of boundary address registers store boundary addresses that partition the address space into a plurality of address ranges corresponding to the plurality of ISA decoding modes.

- 5 The ISA mode selection logic is coupled to the plurality of boundary address registers. The ISA mode selection logic receives the address, and compares the address to determine the ISA decoding mode for the program instruction.

One aspect of the present invention features an ISA  
10 mode selection apparatus in a CPU, where the CPU is configured to execute an application program having program instructions corresponding to one or more ISAs. The ISA mode selection apparatus has a boundary address register file and an ISA mode controller. The boundary address  
15 register file maps ISA modes to address ranges within the CPU's address space. The ISA mode controller is coupled to the boundary address register file. The ISA mode controller designates a specific ISA mode that is to be used to execute a specific program instruction, where the specific program  
20 instruction is located at an address within the CPU's address space. The ISA mode controller includes address evaluation logic that determines a specific address range within which the address lies.

Another aspect of the present invention contemplates a CPU for executing a multiple-ISA program. The CPU includes ISA mode selection logic, ISA mode boundary address registers, and an instruction decoder. The ISA mode selection logic provides a first ISA mode that corresponds to a first program instruction, where the first program instruction is fetched from a first address in memory. The ISA mode boundary address registers are coupled to the ISA mode selection logic. The ISA mode boundary address registers partition the memory into address ranges, where one of a plurality of ISA modes is mapped to each of the address ranges, and where the first address lies within one of the address ranges. The instruction decoder is coupled to the ISA mode selection logic. The instruction decoder receives the first ISA mode, and decodes the first instruction according to the first ISA mode.

Yet another aspect of the present invention provides a computer program product for use with a computing device. The computer program product includes a computer usable medium, having computer readable program code embodied in the medium, for causing a CPU to be described, the CPU being capable of executing a multiple-ISA application program. The computer readable program code includes first program code and second program code. The first program code provides boundary address registers, configured to partition



regard to the following description, and accompanying drawings where:

FIGURE 1 is a diagram illustrating how various components of a related art application program may be generated according to different instruction set architectures, where selection of a particular instruction set architecture for a particular component is based upon desirable characteristics of the particular instruction set architecture.

FIGURE 2 is a block diagram illustrating how a related art multiple-ISA processor decodes and executes an application program consisting of program instructions taken from three different instruction set architectures.

FIGURE 3 is a diagram illustrating present day techniques that are used by related art processors to select ISA decoding modes when executing multiple-ISA application programs.

FIGURE 4 is a block diagram of a portion of a multiple-ISA processor according to the present invention having a boundary address register file for selection of ISA modes.

FIGURE 5 is a block diagram illustrating pipeline stages of a multiple-ISA processor according to the present invention.

FIGURE 6 is a block diagram depicting ISA mode selection logic within the decode/register stage of the processor shown in FIGURE 5.

FIGURE 7 is a flow chart illustrating a method according to the present invention for encoding and executing components of a multiple-ISA application program.

#### DETAILED DESCRIPTION

In light of the above background on the techniques used by present day CPUs to switch between different ISA modes during the execution of a multiple-ISA application program, several related art examples will now be discussed with reference to FIGURES 1-3. These examples point out the problems associated with developing and executing multiple-ISA application programs for execution by today's processors. More particularly, present day multi-ISA programming/execution techniques either partition a processor's address space into fixed and equal-sized segments, or they preclude an individual component (i.e., module, subroutine, task, etc.) of a multiple-ISA application program from being changed from one ISA to the next, without necessitating that all components (i.e., both subordinate and dominant components) referenced by the individual component be modified as well. Following this

discussion, a detailed description of the present invention will be provided with reference to FIGURES 4 through 7. The present invention prevails over the limitations of present day multi-ISA approaches by providing an apparatus and method for selecting ISA decode/execution modes in a CPU in accordance with a set of address boundaries stored in an internal register file, thereby allowing ISA decode/execution modes for program instructions to be selected based solely upon the location in memory of a program instruction. The capability of specifying address boundaries within the register file moreover enables designers to configure variable-sized ISA mode segments within the processor's address space to tailor memory storage requirements for individual program components comprising each of the ISA modes.

Now referring to FIGURE 1, a diagram 100 is presented illustrating how various components 112, 122, 132 of a related art application program can be generated according to different instruction set architectures (ISAs) 110, 120, 130, where selection of a particular instruction set architecture for a particular component is based upon desirable characteristics of the particular instruction set architecture. The diagram 100 depicts three different instruction set architectures: ISA 1 110, ISA 2 120, and ISA 3 130. In this example, program instructions from any of



At a very high level, an ISA 110, 120, 130 comprises  
5 those features of a central processing unit (CPU) or  
microprocessor that are essential for a designer to know.  
In most instances, those essential features consist of the  
program instructions that are used to develop application  
programs to run on the CPU/microprocessor along with the  
10 architecture of programmable resources within the  
CPU/microprocessor such as register files and special  
purpose functional units (e.g., floating point logic).  
Examples of ISAs that are well known in the art today  
include MIPS32, MIPS64, PowerPC, and x86.

17

instructions can be executed on any processor that conforms to the MIPS32 ISA.

In earlier years, application program designers suffered from the restriction of having to encode all of the components 112, 122, 132 of an application program with program instructions from a single instruction set architecture 110, 120, 130. For example, an industrial control application program developed to execute on a PDP11 CPU comprised program instructions taken only from the PDP11 ISA. Any change in the CPU resulted in a requirement to recode all of the components of the application program using program instructions that conformed to the ISA of the new CPU. Consequently, selecting a specific ISA 110, 120, 130 for use in an application program was generally considered by designers to be at the same priority level as selection of a specific CPU for execution of the application program. CPUs and their corresponding instruction set architectures used to be very tightly coupled.

As technology advanced, system designers noted that a substantial amount of application code could be reused following upgrade of a system's CPU because, although the CPU had changed, the application program requirements 140 had not changed. But the existing application code could not be reused in a practical sense because the application

During the late 1970's, CPU designers began to embrace the concept of minimizing the changes to existing software by providing means for executing old code 112, 122, 132 on a new CPU in addition to providing for the execution of new code 112, 122, 132 on the new CPU. What this means is that provisions were made in a new CPU design to implement an older ISA 110, 120, 130 in addition to providing a newer ISA 110, 120, 130. One skilled in the art will remember that Digital Equipment Corporation's VAX11 CPUs provided a capability to execute applications written in program

instructions according to 1) the newer VAX ISA, or 2) the older PDP11 ISA.

In more recent years, however, CPUs have been developed that are capable of non-exclusively executing application programs consisting of program instructions taken from more than one ISA 110, 120, 130. The capability to execute a multiple-ISA application program is a very powerful feature because it provides application program designers with the flexibility to select a specific ISA 110, 120, 130 to implement specific requirements 140 of an application program that exploits desirable characteristics of the specific ISA 110, 120, 130. FIGURE 1 shows an exemplary set of application program requirements 140 that are effectively implemented into a multiple-ISA application program consisting of program instructions taken from three ISAs 110, 120, 130, where each of the three ISAs 110, 120, 130 possess different desirable properties.

In this example, program instructions and resources according to ISA 1 110 are optimized for fast execution on a conforming CPU, however, ISA 1 program instructions are long and require a lot of memory to store. Program instructions and resources according to ISA 2 120 are optimized to require a small amount of memory, but execution of ISA 2 encoded functions on a conforming CPU is much slower than

execution of the same functions when encoded using ISA 1 110 instructions. Program instructions and resources according to ISA 3 130 are optimized to implement certain special functions (e.g., Fast Fourier Transform), yet other  
5 functions encoded by ISA 3 instructions require a lot of storage space and execute much slower than they would were they to be encoded by instructions from ISA 1 110 or ISA 2 120.

The set of requirements 140 for the application program  
10 of FIGURE 1 depicts three general categories of functions: special functions, that are most effectively implemented using ISA 3 program instructions; initialization and operating system functions, that typically must exhibit low latencies and are therefore most effectively encoded using  
15 program instructions from ISA 1; and a number of remaining general purpose functions that neither require special instructions nor fast execution. The general purpose functions could perhaps be encoded using ISA 1 instructions, but in a system configuration that is memory constrained, a  
20 better approach would be to implement all of the general purpose functions using program instructions taken from ISA 2 120.

Hence, a multiple-ISA application program that satisfies the program requirements 140 shown in FIGURE 1 is



the memory 220 in an order that is prescribed by the application program itself. Generally speaking, the fetch logic 212 retrieves a particular instruction 222, 224, 226 from a particular address in the memory 220. The particular  
5 program instruction is provided to the mode switch/decode logic 214. The mode switch/decode logic 214 decodes the particular program instruction into control words or control signals (not shown) that direct the execution logic 216 to perform an operation prescribed by the particular program  
10 instruction. The execution logic 216 receives the control words/signals and, in turn, performs the prescribed operation. Virtually all present day processors 210 fetch program instructions 222, 224, 226 from memory 220 in sequentially ascending or sequentially descending address  
15 order. Changes in control flow of the application program are achieved through the use of control flow modification instructions, generally referred to in the art as jump instructions. Accordingly, during execution of the application program, the fetch logic 212 continues to  
20 generate sequential addresses for the retrieval of sequential program instructions 222, 224, 226 until a jump instruction is encountered. Usually, the jump instruction prescribes a target address in memory 220 that contains the next instruction to be executed following the jump  
25 instruction.

As alluded to above, the primary function performed by the mode switch/decode logic 214 is translation of a program instruction 222, 224, 226 fetched from memory 220 into associated control words/signals that direct the execution logic 216 to perform a corresponding prescribed operation. This translation, or decoding, of program instructions 222, 224, 226 is an extremely complex task that is very closely tied to the architecture of the CPU 210. If the CPU 210 is capable of implementing, or emulating, more than one ISA, then the complexity of instruction decoding becomes more complex. For example, an ISA 1 instruction 222 stored in memory 220 may very well have the same bit states as an ISA 2 instruction 224. But even though these two instructions 222, 224 are equivalent in value to the observer, because they correspond to two entirely different instruction set architectures, the two instructions 222, 224 most likely will direct the execution logic 216 to perform two entirely different operations. Decoding rules are different for each different ISA.









33

According to the mode switch technique employed by CPU 310, an instruction 313, SETPSW MODE2, is first executed which directs the second CPU 320 to set a mode bit 335 within the program status word 334, thus signaling the CPU 330 to switch to ISA mode 2. An ISA 2 jump instruction 313, JMP Y, follows in the sequence that directs the CPU to transfer program control to address Y. The use of a bit 335 or bits within the program status word 334 to accomplish ISA mode switches is described by Jaggar in U.S. Patent number 5,568,646 and U.S. Patent number 5,740,461. Accordingly, during execution of component A 311, ISA 1 instructions 312 are fetched by CPU 330 and a multi-ISA decoder 332 monitors the state of the mode bit 335 to determine which ISA decoding rules to apply for a current instruction 324. When the instruction

AS

25

to transfer program control to address Y. In particular, a bit 345 or bits of a program counter register 344 are employed to indicate which ISA mode is to be used by a multi-ISA decoder 342. Like the technique used by CPU 330, 5 the decoder 342 of CPU 340 monitors the state of bit 345 maintained in program counter register 344 to determine which ISA mode is to be used. Nevill describes this approach for mode switching in U.S. Patent number 5,578,115 and U.S. Patent number 6,021,265. Nevill refers to the type 10 of instruction 313 that modifies the contents of the program counter register 344 to direct a mode switch as a "veneer" 313. According to Nevill, the bit 345 or bits that are employed to signal the decoder 342 to switch modes are either not provided to its memory system or the system is 15 configured to ignore such signaling information. Accordingly, during execution of component A 311, ISA 1 instructions 312 are fetched by the CPU 340 and provided to a multi-ISA decoder 342. When the ISA 1 veneer 313 is executed, the decoder 342 detects state of the bit 345 and 20 switches to ISA 2 mode. It is noted that according to Nevill's scheme, jump target addresses must be manipulated in a calling routine 311 to ensure proper decoding of instructions 316 in a called routine 315. Hence, according to the third technique, the calling component 311 must 25 ensure that the contents of the program counter register 344

are manipulated to properly indicate the ISA mode of the called component 315. One skilled in the art will appreciate as well that manipulation of the mode switch bit 345 in the program counter register 344 by a first ISA 2 instruction 316 in the called component 315 would not be recommended for the same reasons as put forth in the discussion with reference to the program status word technique.

It is significant to note that under any of the mode switching techniques illustrated by the examples of FIGURE 3, it is impossible to independently generate program components 311, 315 in a multiple-ISA application program. In all cases a transferring component 311 must have knowledge of the ISA mode of a transferred component 315 because a mode switch is accomplished by programming a mode switch instruction 313 within the instruction flow of the transferring component 311. As a result, if a designer desires to recode any component of an application program using instructions from a different ISA, then all of the components that are referenced by that component must be modified as well. This is a problem that cuts against the grain of one of the major objectives within the software engineering community, that is, to minimize the number of changes that are required when an application program is modified for reuse. More specifically, when one program

15

20

25

component is encoded into a different ISA, changes are also required to be made in all components that are referenced by the program component in order to modify mode switch instructions so that they indicate the different ISA mode.

5 The multi-ISA techniques discussed above open the door for errors to enter into a system design. One skilled in the art will agree that it is desirable to change only those components of an application program that truly require modifications.

Larson, in U.S. Patent number 5,115,500, advocated an approach for providing independent program components in a multiple-ISA application program by using the uppermost bits of a program instruction's address as means for signaling the ISA mode of the program instruction. In the specific embodiment described by Larson, the upper three address bits were used to determine one of two (or more) ISA decoding modes. Program components encoded according to, say, ISA 1 mode, were to be stored in a first one of eight memory segments, program components encoded according to ISA 2 mode were stored in the remaining segments (in accordance with one embodiment).

Although the technique described by Larson is desirable from the standpoint that program components are effectively decoupled from all other referenced program components,



AN

15



A8 418, and ISA N 419 stored in their respective address ranges in memory 410.

Operationally, the address space, or memory range, of the CPU 450 according to the present invention is partitioned according to the contents of the boundary address registers 461. In the embodiment shown in FIGURE 4, a default value of address 0 provides a lower bound for the address range corresponding to ISA 1 mode. Register BAR 0 461 provides the lower bound, BDY2, corresponding to ISA 2 mode. Hence, the ISA 1 address range spans from address 0 through address BDY2-1. In an alternative embodiment, an additional register 461 is provided to specifically prescribe the lower bound for the ISA 1 address range. Register BAR 1 461 provides a lower bound for the ISA 3 address range, thus establishing an upper bound (i.e., BDY3-1) on an address range for ISA 2 components.

In the embodiment shown in FIGURE 4, the memory space 410 is partitioned into unequal segments to accommodate the storage requirements of an exemplary multi-ISA application program stored therein. The featured embodiment implicitly maps ISA modes to the index of a particular boundary address register 461. For example, if an instruction's address falls within the address range bounded by the contents of BAR 0 461 and BAR 1 461, then the ISA decoding mode that is

applied to the instruction is mapped to ISA mode 2. Mapping of a particular ISA mode to a particular boundary address register 461 can be achieved by the register's index, or, in an alternative embodiment, a portion of the contents of the boundary address register 461 comprise a field (not shown) that indicates the particular ISA mode to be used to decode/execute instructions that fall within that address range.

In an embedded application embodiment, contents of the boundary register file 460 are established during initialization of the CPU 450 via hardwired signals (not shown) or via the execution of code from a boot read-only memory (ROM) (not shown). In a non-embedded embodiment, contents of the register file 460 can be established either via boot ROM during initialization, or the boundaries can be dynamically altered by an operating system as application programs are fetched and loaded into the memory 410.

Note that both components A 411 and B 415, in contrast to like components 311, 315 discussed with reference to FIGURE 3, do not contain any "mode switch" instructions. This is because mode switch instructions are not required for the processor 450 according to the present invention; ISA mode management is directly mapped to address ranges in the CPU's address space. The ISA 1 instruction 412 that

directs the CPU 450 to transfer program flow to address Y is merely an ISA 1 jump instruction 412. And the ISA 2 instruction 416 that directs the CPU 450 to return program flow to address X is merely an ISA 2 jump instruction 416.

5 Component A 411 is not required to know the ISA mode of any of the components that it references. For example, if a system designer were to recompile component B 415 such that it comprised program instructions according to ISA 3 mode, then component B 415 would be the only component that

10 required changing within the application program. Linker software would then assign the newly encoded component B 415 to the address range corresponding to ISA 3 mode. Hence, the present invention minimizes the number of changes that are required when reusing previously compiled components in

15 a multi-ISA application program.

In operation, the fetch stage 510 fetches program instructions from the memory 560 in an order prescribed by an application program. The address of each fetched program instruction is carried along with the program instruction in an instruction pointer buffer 512. Fetched program instructions and their addresses are provided to the decode/register stage 520.

The decode/register stage 520 decodes a fetched program instruction into control words/signals that direct logic in subsequent stages 530, 540, 550 of the CPU 500 to perform certain subtasks corresponding to an operation prescribed by the fetched program instruction. Additionally, contents of a general purpose register file (not shown) are accessed as prescribed by the program instruction within the decode/register stage. In the embodiment of the present invention shown in FIGURE 5, when the program instruction is provided to the decode/register stage 520, the program instruction's address is received into the mode control logic 524. The mode control logic 524 compares the program instruction's address against the contents of the boundary register file 522 to determine a particular address range within which the address lies. The mode control logic 524 then selects a particular ISA mode that corresponds to a particular boundary address register (not shown) whose contents bound the particular address range. Thus, the

program instruction is decoded and executed according to the particular ISA mode selected by the mode control logic 524.

Control words/signals and the contents of general purpose registers (if any) are provided to the execute stage 530, wherein results of the prescribed operation are generated. The results are provided to the data stage 540.

The data stage 540 executes load and store operations to data memory (not shown). Contents of a general purpose register are written to the data memory as prescribed by logic within this stage 540 or contents of a data memory location can be retrieved and provided to the write back stage 550.

Sub A9  
The write back stage 550 writes the results generated in the execute stage 530 or contents of data memory retrieved by the data stage 540 into prescribed registers in the general purpose register file. Hence, program instructions are fetched from memory 560 by the fetch stage logic 510 and synchronously proceed through subsequent CPU stages 520-550 in a fashion very much like an assembly line. Accordingly, the present invention does not require that any additional "switch" or "veneer" instructions be inserted into the pipeline flow in order to explicitly direct the CPU 500 to switch ISA modes because a given program instruction's ISA mode is implicitly carried in its

A9  
corresponding address. This is advantageous from a execution speed perspective because the insertion of additional instructions into the flow of the pipeline bogs down the execution of an application program

5 The architectural stages 510-550 of the CPU embodiment presented in FIGURE 5 are representative of a multi-ISA CPU. Particular CPUs may have more or less stages, or the functions of a particular CPU may be partitioned differently, or certain functions may appear in a slightly  
10 different order (such as those functions discussed with reference to the execute 530 and data stages 540). Regardless of the variations that exist, however, one skilled in the art will appreciate that the ISA mode selection logic 524 must be within or precede the decode  
15 stage 520. The illustrated CPU embodiment 500 stations the mode control logic 524 and the boundary register file 522 within the decode/register stage 520 because other general purpose registers are accessed within this stage 520 as well.

20 Now referring to FIGURE 6, a block diagram is presented depicting ISA mode selection logic 620 within a decode/register stage 600 of the processor 500 shown in FIGURE 5. The decode/register stage 600 has instruction decode logic 640 that receives a program instruction from a



program instruction register 610. The register 610 has an instruction field 611 for the program instruction itself (i.e., binary representation of the instruction including, for example, opcode) and an address field 612 that contains the address of the program instruction. Contents of the instruction field 611 are provided to the instruction decoder 640 and contents of the address field 612 are provided to the ISA mode selection logic 620. The ISA mode selection logic 620 includes address evaluation logic 621 that is coupled to a boundary address register file 630. The ISA mode controller 620 provides an ISA mode output via bus 622 to the instruction decoder 640. The instruction decoder 640 outputs decoded control words/signals to subsequent CPU stages (not shown) via an execution control register 642. Exemplary ISA mode address range boundaries are shown loaded within boundary address registers BAR 1 631 through BAR 7 631.

Operationally, as a program instruction flows from fetch stage logic (not shown) to the decode/register stage 600, its address is retrieved by the address evaluation logic 621 from the address field 612 of the instruction buffer 610. The address evaluation logic 621 compares the retrieved address against the address ranges defined by the contents of the boundary address registers 631 in the register file 630. In one embodiment, the address

evaluation logic 621 sequentially compares the retrieved address to the contents of the registers 631 to determine the particular address range within which the retrieved address lies. In an alternative embodiment, the address

5 evaluation logic 621 performs parallel comparisons to determine the particular address range. As FIGURE 6 depicts, retrieved address D0000000 falls within the particular address range bounded by addresses 0C0000000 and 0E0000000 prescribed respectively by boundary address

10 registers BAR 1 631 and BAR 2 631. In a lower address bound embodiment, the retrieved address of the program instruction is mapped to register BAR 1 631. The address evaluation logic 621 confirms to the ISA mode controller 620 that address D0000000 corresponds to boundary address register

15 BAR 1. The mode selector 620, in turn, outputs ISA mode 1 over bus 622.

Accordingly, the instruction decoder 640 implements decoding rules according to ISA mode 1 to correctly decode and execute the ISA 1 program instruction provided in the

20 instruction field 611 of the instruction buffer 610. The program instruction's correctly decoded control words/signals are thus output to the execution control register 642.

Now referring to FIGURE 7, a flow chart 700 is presented illustrating a method according to the present invention for encoding and executing components of a multiple-ISA application program.

5       Flow begins at block 702, where compiled components of a multi-ISA application program are provided to a linker/loader application program according to the present invention. Flow then proceeds to block 704.

At block 704, software within the linker/loader program  
10       processes the components of the multi-ISA application program. The linker/loader segregates components of the program into categories corresponding to each one of a plurality of ISA modes that are employed within the application program. The distribution of address space  
15       required among all of the components falling into each one of the ISA modes is used by the linker/loader to determine and establish address ranges in the address space of a CPU according to the present invention. Each of the address ranges is mapped to an address boundary that is to be stored  
20       in a corresponding address boundary register within the CPU. Flow then proceeds to block 706.

At block 706, the linker/loader loads contents of the boundary address registers and all of the program components into their corresponding address range in a memory device

Sub  
A10

At block  
invention fet  
program from  
Along with th  
instruction, A

10

15

At block 712, ISA mode selection logic in the CPU selects a specific ISA decoding mode for the next instruction that equals the boundary register index determined in block 710. Flow then proceeds to block 714.

20

708, where an instruction following the next instruction is fetched from memory.

The method continues until the CPU ceases fetching instructions, an event that is typically caused by removal  
5 of power.

The examples of FIGURES 4 through 7 clearly illustrate that a multi-ISA application program can be effectively executed on a CPU according to the present invention without requiring complex cosmetic interrelationships between  
10 calling and called program components. This is because the present invention allows a program instruction's ISA mode to be established by its address within the CPU's address space. When a designer desires to change the ISA mode of a given component within the application program, all that is  
15 required is that the given component be recoded into the chosen ISA mode; no changes are required to be made to components that call the given component or to components called by the given component. Moreover, address ranges corresponding to different ISA modes can be flexibly  
20 tailored to serve differing ISA mode storage requirements of the program because address range boundaries are based upon the contents of a boundary address register file that is loaded prior to or at the time of execution of the application program.

Although the present invention and its objects, features, and advantages have been described in detail, other embodiments are encompassed by the invention as well. In addition to implementations of the invention using  
5 hardware, the invention can be embodied in software disposed, for example, in a computer usable (e.g., readable) medium configured to store the software (i.e., computer readable program code). The program code causes the enablement of the functions or fabrication, or both, of the  
10 invention disclosed herein. For example, this can be accomplished through the use of general programming languages (e.g., C, C++, etc.), hardware description languages (HDL) including Verilog HDL, VHDL, AHDL (Altera Hardware Description Language) and so on, or other  
15 programming and/or circuit (i.e., schematic) capture tools available in the art. The program code can be disposed in any known computer usable medium including semiconductor memory, magnetic disk, optical disc (e.g., CD-ROM, DVD-ROM, etc.) and as a computer data signal embodied in a computer  
20 usable (e.g., readable) transmission medium (e.g., carrier wave or any other medium including digital, optical or analog-based medium). As such, the code can be transmitted over communication networks including the Internet and intranets. It is understood that the functions accomplished  
25 and/or structure provided by the invention as described

above can be represented in a core that is embodied in program code and may be transformed to hardware as part of the production of integrated circuits. Also, the invention may be embodied as a combination of hardware and software.

Sub  
All

5 In addition, the present invention has been particularly characterized in terms of a CPU or microprocessor. In particular, one embodiment of the present invention described with reference to FIGURE 5 portrays application its application within a 5-stage  
10 pipelined CPU 500. These specific embodiments and characterizations are presented herein as representative embodiments for the present invention, however, such description should by no means restrict application of the concept of basing ISA decoding mode for the processing of  
15 program instructions upon prescribed and variable-sized address ranges. On the contrary, the present invention can be embodied within a multi-ISA graphics processor, a multi-ISA digital signal processor, as well as less commonly known components to include multi-ISA communications processors,  
20 multi-ISA video processors, multi-ISA memory controllers, and multi-ISA micro controllers.

Furthermore, the present invention has been specifically presented in terms of a multiple-ISA CPU that is capable of implementing certain well-known instruction

set architectures to include MIPS32, MIPS64, x86, and PowerPC. These exemplary ISAs are employed herein because they provide a recognizable basis for teaching the present invention, however, it should not be construed that application of the present invention is limited to these ISAs. Rather, the present invention contemplates boundary address-based ISA mode distinction of program instructions included in instruction set extensions within a family of instructions such as MIPS32, MIPS64, 16/32-bit x86, MMX, etc., as well as distinctions between the ISAs of different manufacturers.

Finally, CPU embodiments according to the present invention have been described at a level that does not rely upon the type of instruction sets employed, how the instructions are formatted, or how the instructions are processed within the CPU. This is because address-based ISA mode selection contemplates application within complex instruction set architectures (CISC), reduced instruction set architectures (RISC), architectures providing for fixed-length or variable-length instructions, in-order processors, and out-of-order processors as well as the embodiments specifically described herein.

Those skilled in the art should appreciate that they can readily use the disclosed conception and specific



What is claimed is: